@[TOC](Mit6.S081-实验3-Page tables)

# 一、Print a page table

## 1，实验准备

1）阅读xv6 book章节3

2）内存布局代码：kern/memlayout.h

3）虚拟内存代码：kernel/vm.c

4）分配、释放物理内存代码：kernel/kalloc.c

## 2，实验要求

为了帮助学习RISC-V page tables，也可能是帮助未来的调试，第一个任务是写一个函数来打印页表内容。

定义一个函数vmprint()。它应该接收一个pagetable\_t参数，并且用下面格式打印页表。

在exec.c中return argc之前，插入if(p->pid==1) vmprint(p-pagetable)，来打印第一个进程的页表。

当启动xv6时，应该打印以下内容，描述第一个进程的页表信息，在刚执行完exec()时。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110184546890.png#pic\_center)

第一行显示了vmprint()的参数。在那之后每行对应一个PTE，包含树中指向page-table pages的PTE。

每个PTE行由一些".."（表明树的深度）缩进。

每个PTE行显示PTE在page-table page中的索引、pte地址、pte中的物理页地址。不要打印无效PTE。

在上面的例子中，有顶级page-table page：0、255。entry 0的下一级仅有索引0， 这个索引0的下一级有0、1、2三个PTE。

你的代码可能生成与上面那些代码不同的物理地址。entry和虚拟地址的数量应该一样。

## 3，具体实现

1）在kernel/vm.c中定义

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110182746460.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

2）修改kernel/defs.h中增加vmprint()的声明

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110182917947.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

3）在kernel/exec.c中增加对vmprint()的调用

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110183050650.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 4，执行效果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110183139956.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 5，测试效果

在xv6-labs-2020中，执行下面指令，测试程序

```bash

make grade

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201110183348468.png#pic\_center)

# 二、A kernel page table per process

## 1，实验要求

无论何时在内核执行时，xv6使用同一个内核页表。内核页表是一个物理地址的直接映射，因此内核虚拟地址x对应物理地址x。

xv6也有一个单独的页表给每个进程的用户地址空间，仅包含那个进程用户内存的映射，起始于虚拟地址0。

因为内核页表不包含这些映射，用户地址在内核无效。因此，当内核需要使用一个用户指针传到system call时，内核必须首先翻译指针到物理地址。

这个和下个实验的目的是为了允许内核直接解析用户指针。

第一个任务是更改内核，为了当在内核执行时，每个进程使用它自己的内核页表拷贝。

更改struct proc来让每个进程保持一个内核页表，更改scheduler()，当切换进程时切换内核页表。

对于这一步，每个进程的内核页表应该和已存在的全局内核页表完全相同。

读xv6书、理解作业一开始提到的代码如何起作用，将更容易正确地更改虚拟内存代码。

页表设置中的bug会因为缺少映射导致缺陷，导致加载、存储会影响到不可预期的物理内存页，也会导致执行不正确的物理内存页。

## 2，具体实现

1）在kernel/proc.h的struct proc中新加kernelPageTable

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111102954737.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

2）修改kernel/vm.c，新增一个vmmake()方法可以创建一个内核页表（不包含CLINT的映射）

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111103228867.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

3）在kernel/vm.c，修改vminit()方法，内部由vmmake()实现，此处为全局内核页表创建过程，另外加上CLINT的映射。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111103505752.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

4）在kernel/proc.c，修改procinit()方法，不再于此方法中为每个进程分配内核栈

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020111110385910.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

5）在kernel/proc.c，修改allocproc()，在此时创建内核页表，并在内核页表上分配一个内核栈

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020111110414210.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

6）在kernel/proc.c，修改scheduler()，在swtch()切换进程前修改satp，保证进程执行期间用的是进程内核页表，切换完后再修改satp为全局内核页表

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111105022352.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

7）在kernel/vm.c，kvmpa()方法会在进程执行期间调用，此时需要修改为获取进程内核页表，而不是全局内核页表

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111105501451.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

myproc()方法调用，需要在proc.c头部添加头文件引用

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111105638483.png#pic\_center)

8）在kernel/proc.c，修改freeproc()方法，添加对进程内核页表的资源释放

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111105945232.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

8）在kernel/proc.c，新增proc\_free\_kernel\_pagetable()方法，用于释放进程内核页表指向的物理内存，以及进程内核页表本身

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111110246807.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 3，测试效果

启动xv6，成功执行usertests

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020111111055949.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

# 三、Simplify copyin/copyinstr

## 1，实验要求

内核的copyin函数读取用户指针指向的内存。它先将它们翻译为物理地址（内核可以直接用）。通过代码walk进程页表实现翻译。

在此实验中，你的工作是给每个进程的内核页表添加用户映射，使得copyin可以直接使用用户指针。

## 2，具体实现

1）在kernel/proc.c中，修改fork方法

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111112110982.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

2）在kernel/exec.c中，修改exec()，在用户进程页表重新生成完后，取消进程内核页表之前的映射，在进程内核页表，建立新进程页表的映射

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111112401759.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

并添加用户空间地址不能大于PLIC的判断

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111112730288.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

3）在sysproc.c中，修改sys\_sbrk()，在内存扩张、缩小时，相应更改进程内核页表

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111112926707.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

4）在kernel/defs.h中，添加copyin\_new()、copyinstr\_new()的声明

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111113427493.png#pic\_center)

5）在kernel/vm.c中，替换copyin()、copyinstr()为copyin\_new()、copyinstr\_new()

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111113515505.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 3，测试结果

在xv6-labs-2020下执行make grade

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201111114226625.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)